Project on Vending Machine (VM) design using MDA
(Final Report)

by

Pramoda Chandru (A20528337)

Submitted to:
Dr. Bogdan Korel
Department of Computer Science
Illinois Institute of Technology

Submitted Date: 27-April-2023

Introduction:

The goal of this project is to design two different Vending Machine (VM) components using the Model- Driven Architecture (MDA) and then implement these Vending Machine components based on this design.

Description of the Project:
There are two Vending Machine components: VM-1 and VM-2. VM-1 component supports the following operations

VM-1 component supports the following operations:
create(int p); // starts a vending machine application, where p is an initial price of a drink
coin(float v); // a coin with value v is inserted
sugar(); // sugar button is pressed
tea(); // tea button is pressed
latte(); // latte button is pressed
insert_cups(int n); // n cups are inserted into the vending machine
set_price(float p); // new price of a cup of tea/latte is set to value p
cancel(); // cancel selection for a cup of tea or latte

VM-2 component supports the following operations:
CREATE(float p); // starts a vending machine application, where p is an initial price of a drink COIN(int); // a coin with value v is inserted
CARD(int x) // credit card is swiped, where x is an available fund
SUGAR(); // sugar button is pressed
CREAM(); // cream button is pressed
COFFEE(); // coffee button is pressed
InsertCups(int n) // n cups are inserted into the vending machine
SetPrice(int p); // new price of a cup of coffee is set to value p
CANCEL(); // cancel selection for a cup of coffee

Both Vending Machine components are state-based components that control simple vending machines. These vending machines dispose of a cup of tea/coffee/latte with or without additives (sugar and/or cream).

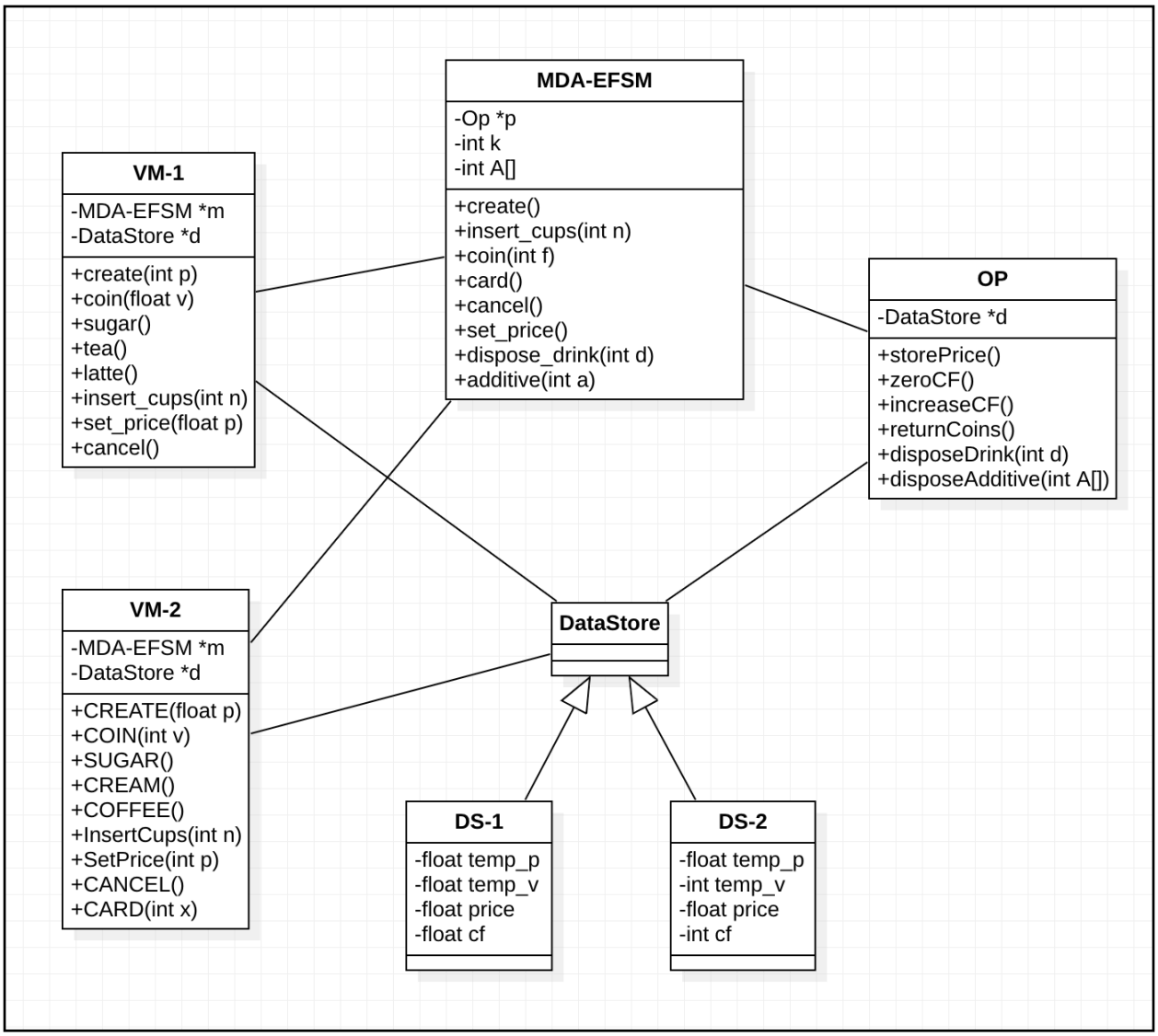
Aspects that vary between the two Vending Machine components:
A. Different methods of payment
B. Different drinks disposed
C. Different additives offered
D. Different messages displayed
E. Different operation names and signatures
F. Different data types

The goal of this project is to design two VM components using the Model-Driven Architecture (MDA) covered in the course. An executable meta-model referred to as MDA-EFSM of VM components should capture the “generic behavior” of two VM components and should be de-coupled from data and implementation details.

This document consists of the MDA-EFSM model report for the VM components with the below topics:

- A class diagram
- A list of meta events for the MDA-EFSM
- A list of meta actions for the MDA-EFSM, where the responsibility of each action must be described
- A state diagram/model of the MDA-EFSM
- Pseudo-code of all operations of Input Processors of VM-1 and VM-2

Class diagram:



List of meta events for the MDA-EFSM:

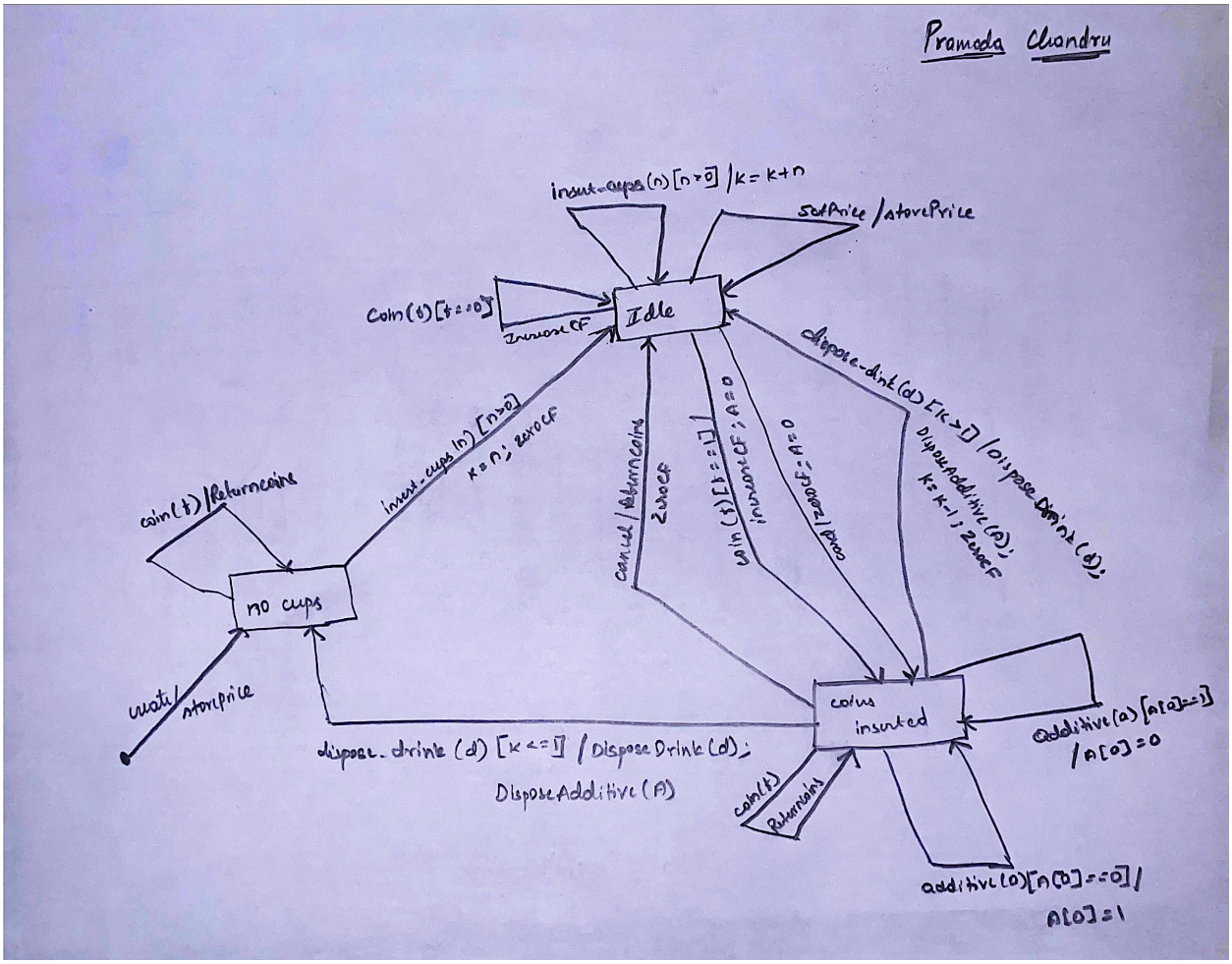
1. create()
2. insert_cups(int n) // n represents # of cups
3. coin(int f) // f=1: sufficient funds inserted for a drink
   // f=0: not sufficient funds for a drink
4. card()
5. cancel()
6. set_price()
7. dispose_drink(ind d) // d represents a drink id
8. additive(int a) //a represents additive id

List of meta actions for the MDA-EFSM:

1. storePrice()
2. zeroCF() // zero Cumulative Fund cf
3. IncreaseCF() // increase Cumulative Fund cf
4. returnCoins() // return coins inserted for a drink
5. dispenseDrink(ind d) // dispose a drink with d id
6. dispenseAdditive(int a) //dispose of marked additives in A list,
   // where additive with i id is disposed when A[i]=1

Part 1:

State diagram/model of the MDA-EFSM:



Below are the new variable introduced:
int A[] // a list of additives to be disposed
int k // keeps track of # of cups Internal Variables

Pseudo-code of all operations of Input Processors

Operations of the Input Processor (VM-1)

```
create(int p) {
  d.temp_p = p;
  m.create();
}

coin(float v) {
  d.temp_v = v;
  if (d.cf+v >= d.price)
    m.coin(1);
  else
    m.coin(0);
}

sugar() {
  d.additive(1);
}

tea() {
  m.dispose_drink(1);
}

latte() {
  m.dispose_drink(2);
}

insert_cups(int n) {
     m.insert_cups(n);
}

set_price(float p) {
   d.temp_p = p;
   m.set_price();
}

cancel() {
  m.cancel();
}
```

Note:
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store -1 object

Operations of the Input Processor (VM-2)

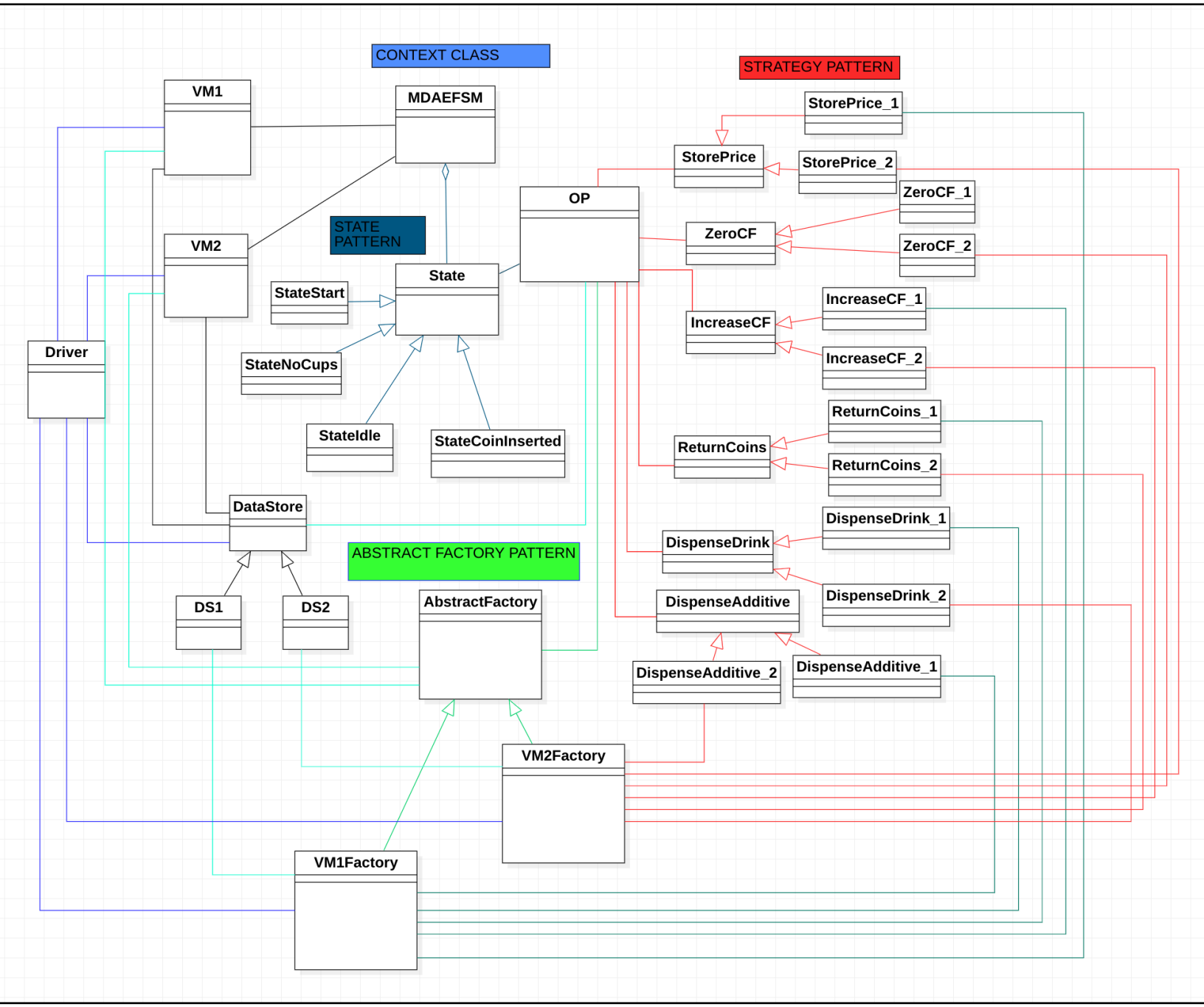
```
CREATE(float p) {
  d.temp_p = p;
  m.create();
}

COIN(int v) {
  d.temp_v = v;
  if (d.cf+v >= d.price)
    m.coin(1);
  else
    m.coin(0);
}

CARD(int x) {
  if(x >= ds.price)
    m.card()
}

SUGAR() {
  m.additive(2);
}

CREAM() {
  m.additive(1);
}

COFFEE() {
  m.dispose_drink(1);
}

insertCups(int n) {
  m.insert_cups(n);
}

setPrice(int p) {
   d.temp_p = p;
   m.set_price();
}
CANCEL() {
   m.cancel();
}
```

Note:
m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store object

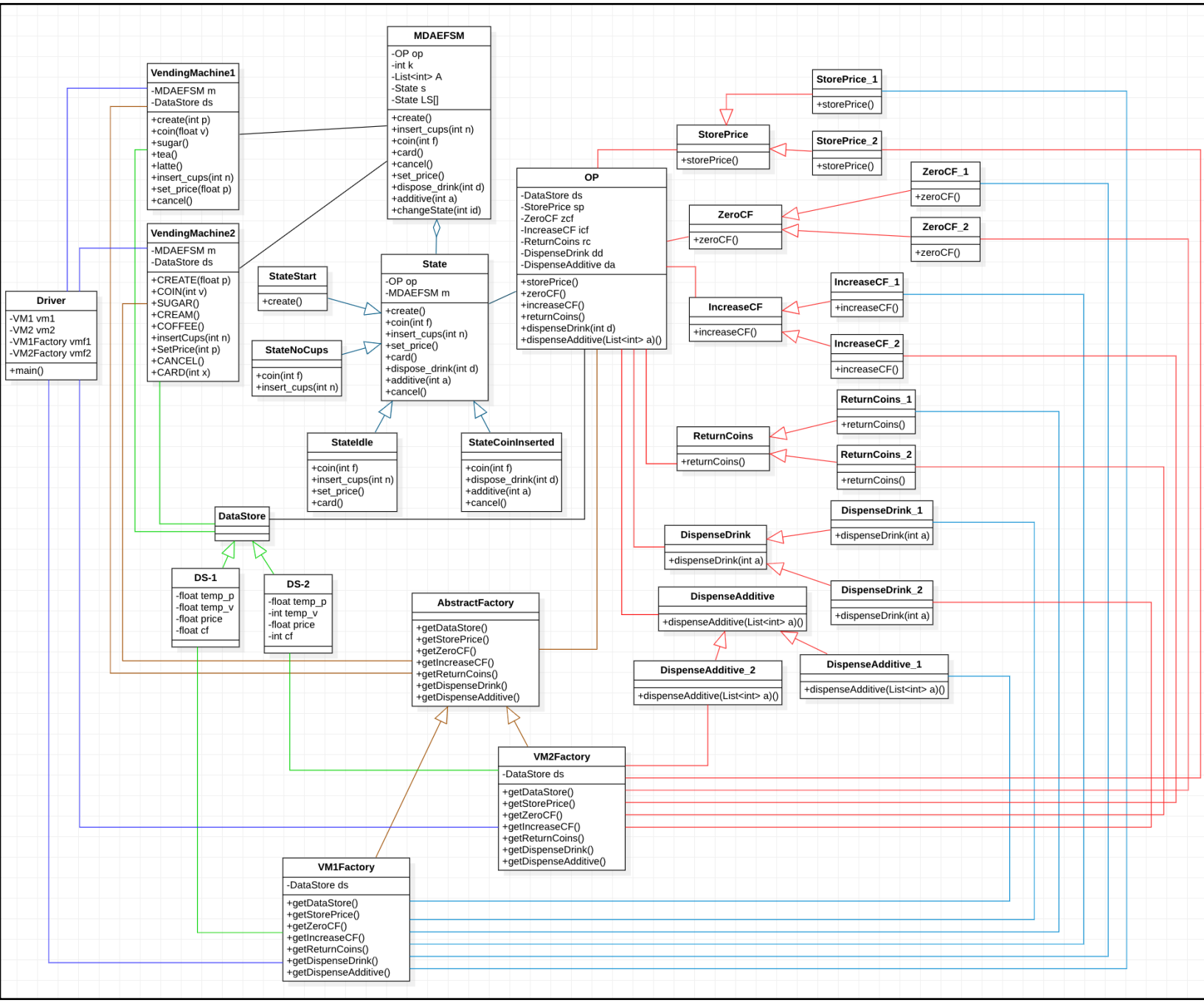
PART 2: Class diagram(s) of the MDA of the VM components.

In this part, we will design two Vending Machines (VM) components using the Model-Driven Architecture (MDA) and then implement these VM components based on this design using the below OO programming language.

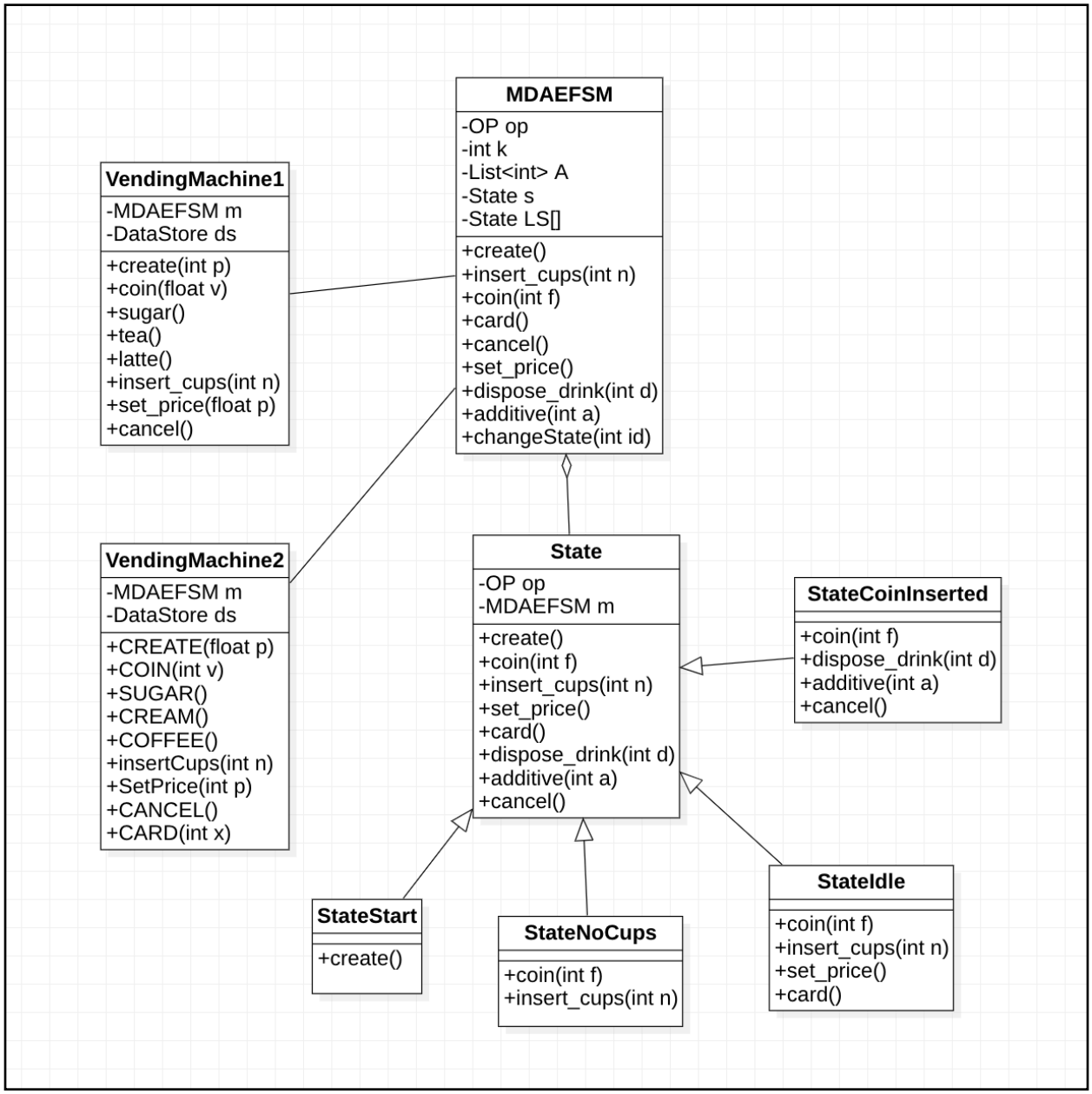
- state pattern
- strategy pattern
- abstract factory pattern
-



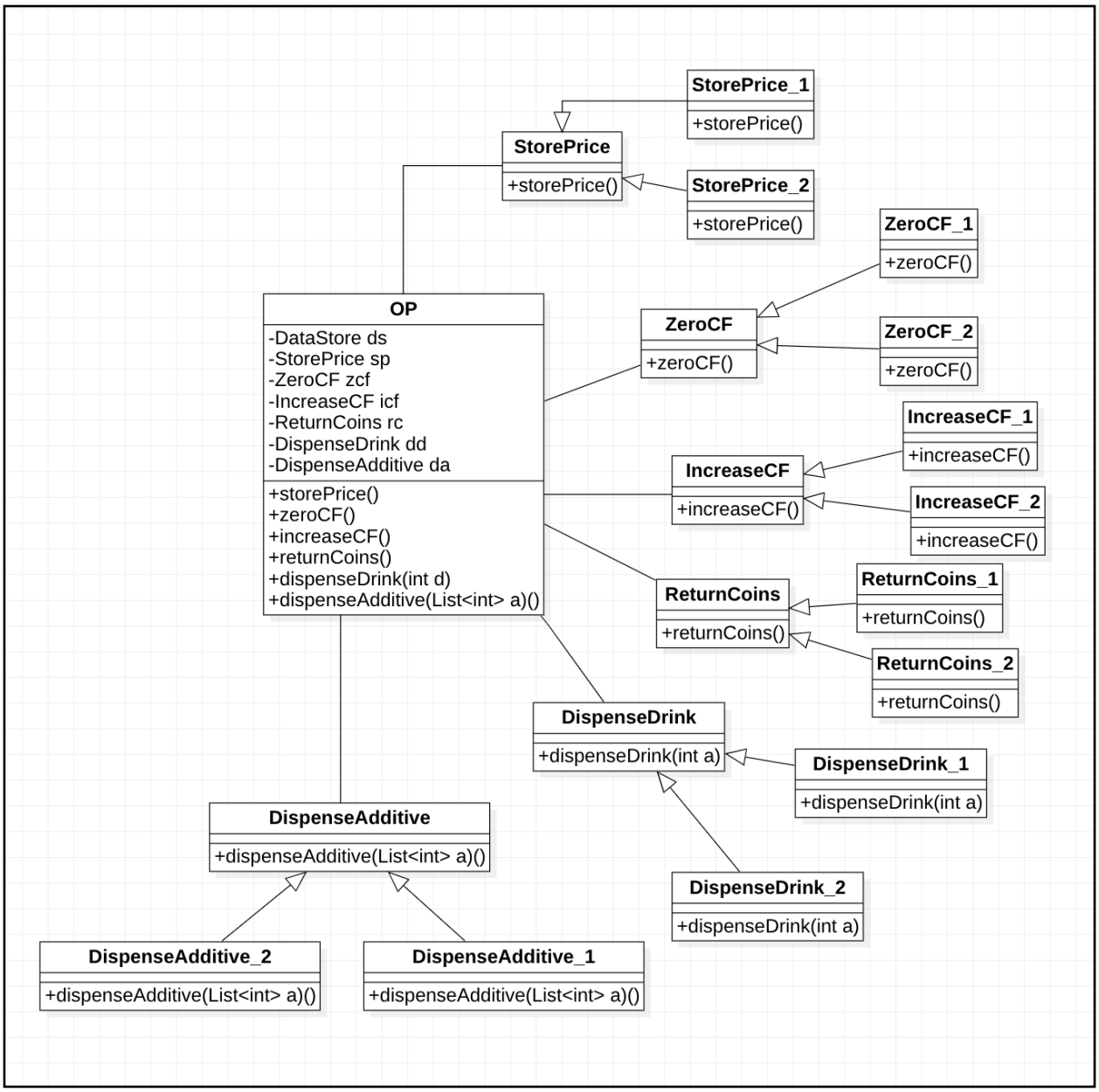
Class Diagram with variable and methods:



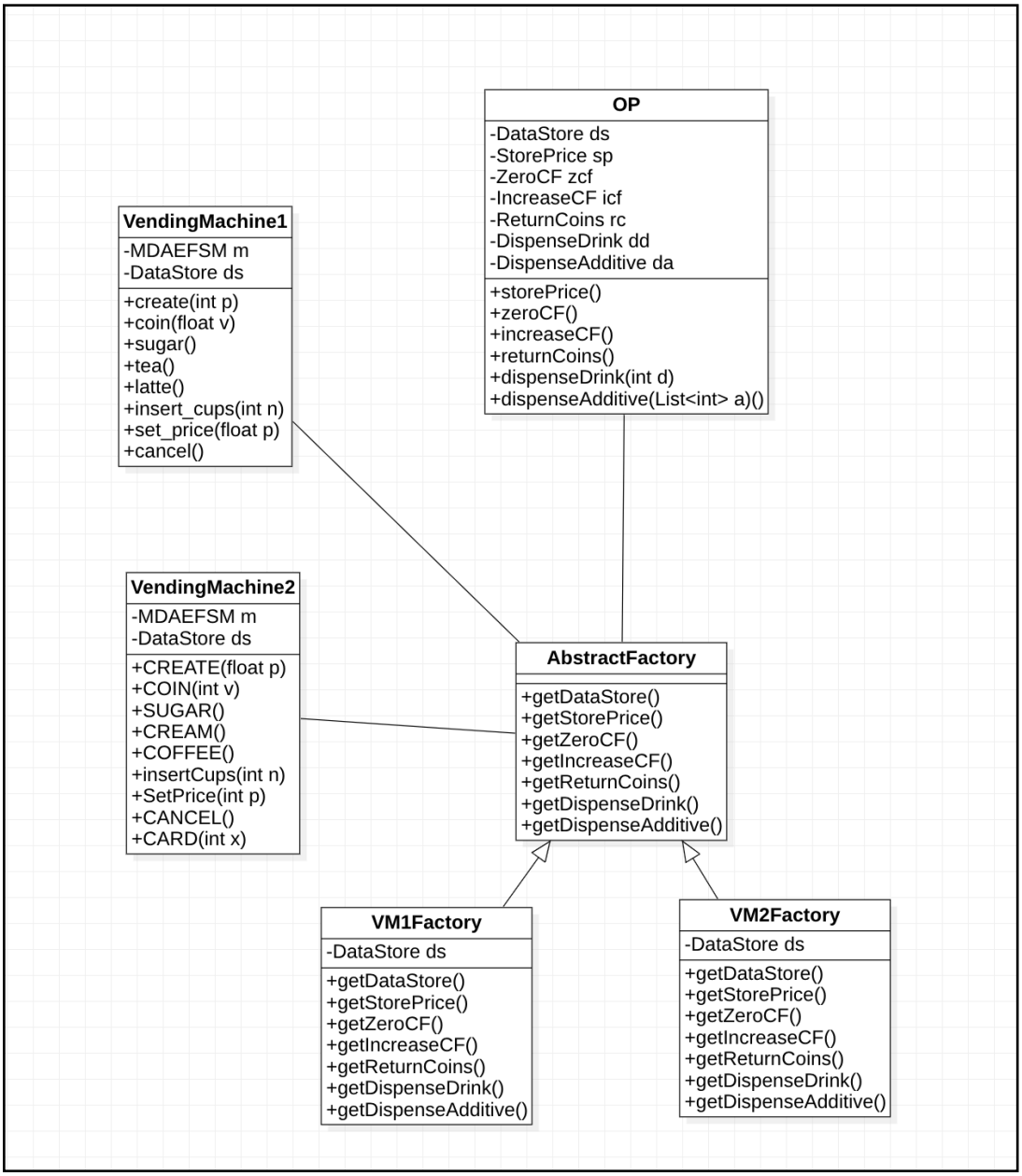
- **State Pattern: State pattern portion of the UlML diagram(Decentralized version is used in the design)**



- **<u>Strategy pattern: Strategy pattern portion of the UlML diagram</u>**



- **<u>Abstract Factory Pattern: Abstract Factory pattern portion of the UlML diagram</u>**



Part 3: Purpose and Responsibilities of each class and their Operation.

Class Name	**Driver.VendingMachineDriver**
Purpose	Main class - Initiate vending machine system execution
Methods	
main()	Execution begins here - user can choose the vending machine and operations

Class Name	**DataStore.DataStore_1**
Purpose	Provides all operations to store data provided in VendingMachine-1
Variables	temp_v, price, cf, temp_p
Methods	Setters and Getters for the above variables to store and retrieve data

Class Name	**DataStore.DataStore_2**
Purpose	Provides all operations to store data provided in VendingMachine-2
Variables	temp_v, price, cf, temp_p
Methods	Setters and Getters for the above variables to store and retrieve data

Class Name	**Factory.VM1Factory**
Purpose	Provides creation of required class objects for the Vending Machine 1 and creates associate datastore
Variables	DataStore of type DataStore1 to hold the reference to the class
Methods	All methods will return the reference to the Vending Machine 1 classes

Class Name	**Factory.VM2Factory**
Purpose	Provides creation of required class objects for the Vending Machine 2 and creates associate datastore
Variables	DataStore of type DataStore2 to hold the reference to the class
Methods	All methods will return the reference to the Vending Machine 2 classes

Class Name	MDAEFSM.MDAEFSM
Purpose	Provides platform independent capabalities to the system and additional systems can easily integrated.
Methods	methods responsible to call the strategy class methods of the respective classes

Class Name	OutputProcesor.OutputProcessor
Purpose	Provides output operation of the MDA and parent class of the all the strategy classes
Variables	S is the reference to current state class SL[] holds the state class list op reference to the output procesor k to hold the number of cup count A[] to hold the additive list
Methods	Setters and Getters for the above variables to store and retrieve data changeState(int id): Changes the reference of the s to the current state Other methods responsible to call the state methods

Class Name	State.State
Purpose	Responsible for providing the bridge between the MDAEFSM and different state classes
Variables	M reference to the MDAEFSM class op referecne to the OutputProcessor class
Methods	All the methods of different state classes are created and displays the default message unless implemented by the state classes

Class Name	State.StateStart
Purpose	Responsible for providing the implementation of the methods responsible and change of state information
Methods	Create - responsible for starting the vending machine Store the intial price calls change of the state to nocups

Class Name	State.StateNoCups
Purpose	Responsible for providing the implementation of the methods responsible and change of state information
Methods	coin - returns the coin as no cups availabel insertCups - initialize the k value with the given input set the CF to 0 change state StateIdle

Class Name	State.StateIdle
Purpose	Responsible for providing the implementation of the methods responsible and change of state information
Methods	coin - accepts the payment in cash and either increase the coin count if payment is insufficient else change the state to coininserted insertCups - Add the number of cups to exisiting cup count set_price - set price of the drink card - accepts the payment in card and change the state to coininserted if sufficient cash availble in the card

Class Name	State.StateCoinInserted
Purpose	Responsible for providing the implementation of the methods responsible and change of state information
Methods	coin - returns coin Cancel - cancel the operation disposeDrink - dispose the selected drink based on the id additive - dispose the selected additive

Class Name	Strategy.DispenseDrink
Purpose	Responsible for dispensing the selected drink
Methods	dispenseDrink - Verifies the id and dispenses the selected drink

Class Name	Strategy.IncreaseCF
Purpose	Responsible for keeping track of the coins inserted
Methods	increaseCF - increaments the coin balance by new value

Class Name	Strategy.ReturnCoins
Purpose	Responsible for returning the coins
Methods	returnCoins - return the inserted coins

Class Name	**Strategy.StorePrice**
Purpose	Responsible for storing the price
Methods	storePrice - store the price of the drink

Class Name	**Strategy.ZeroCF**
Purpose	Responsible for resetting the CF value 0
Methods	zeroCF - reset the CF value to 0

4. Dynamics. Provide two sequence diagrams for two Scenarios:

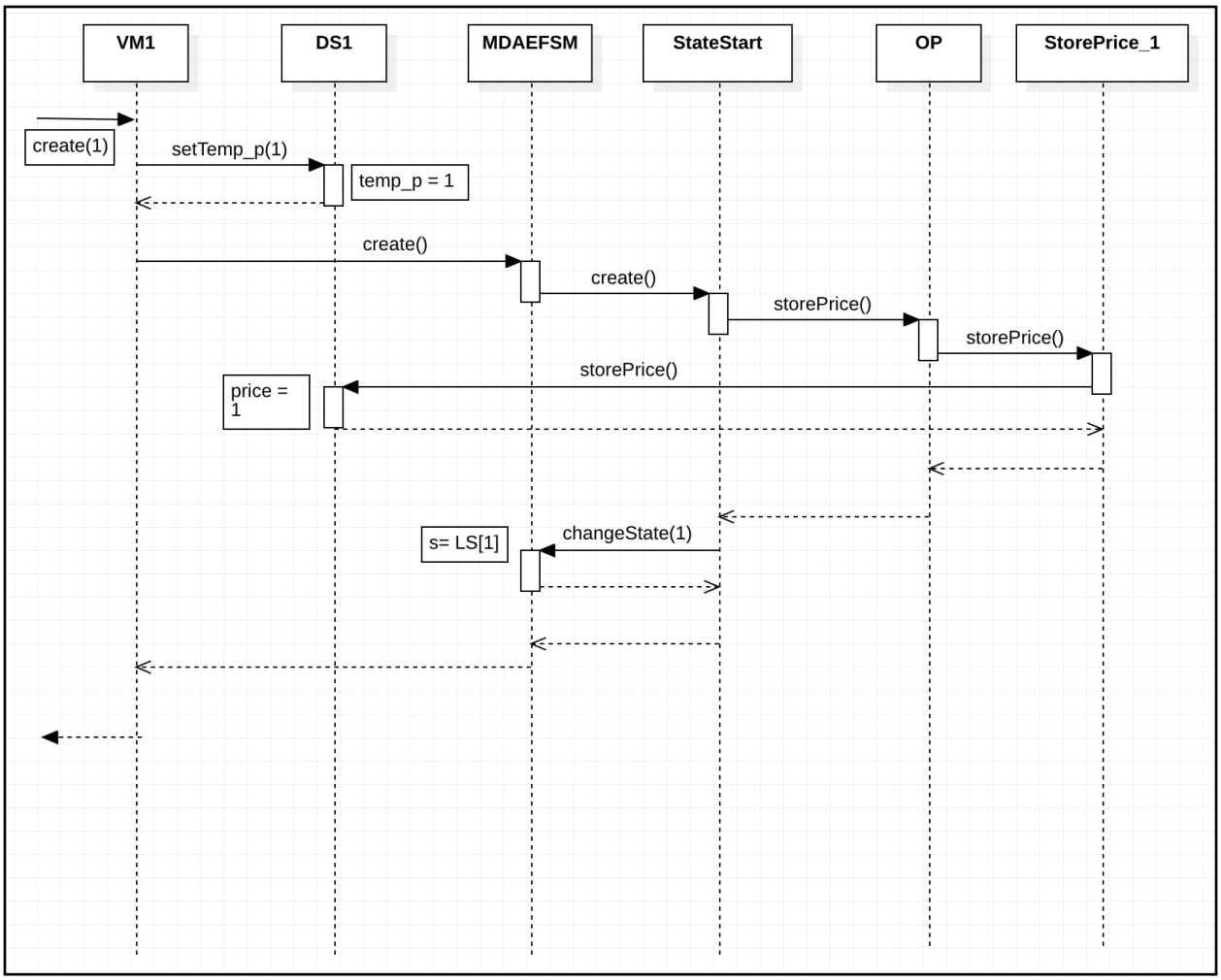
a. Scenario-I should show how the cup of latte is disposed of in the Vending Machine VM-1 component, i.e., the following sequence of operations is issued: create(1), insert_cups(20), coin(0.5), coin(0.5), sugar(), latte()

b. b. Scenario-II should show how a cup of coffee is disposed of in the Vending Machine VM-2 component, i.e., the following sequence of operations is issued: CREATE(1.5), InsertCups(1), CARD(10), CREAM(), COFFEE()

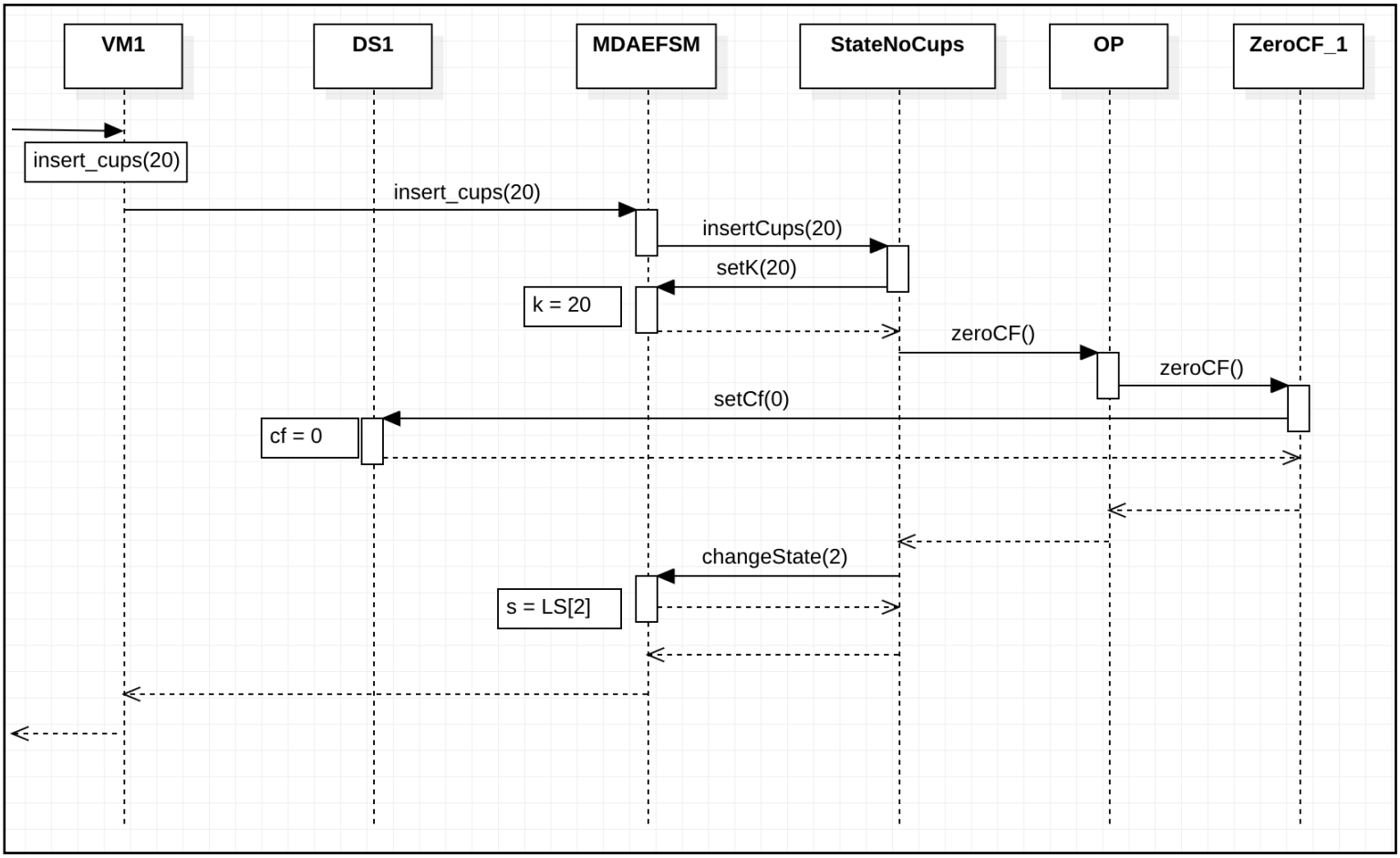
a. Vending machine 1 system sequence diagrams:

Note: Individual operations of the sequence diagram are added below. 1 single sequence diagram was hard to read.

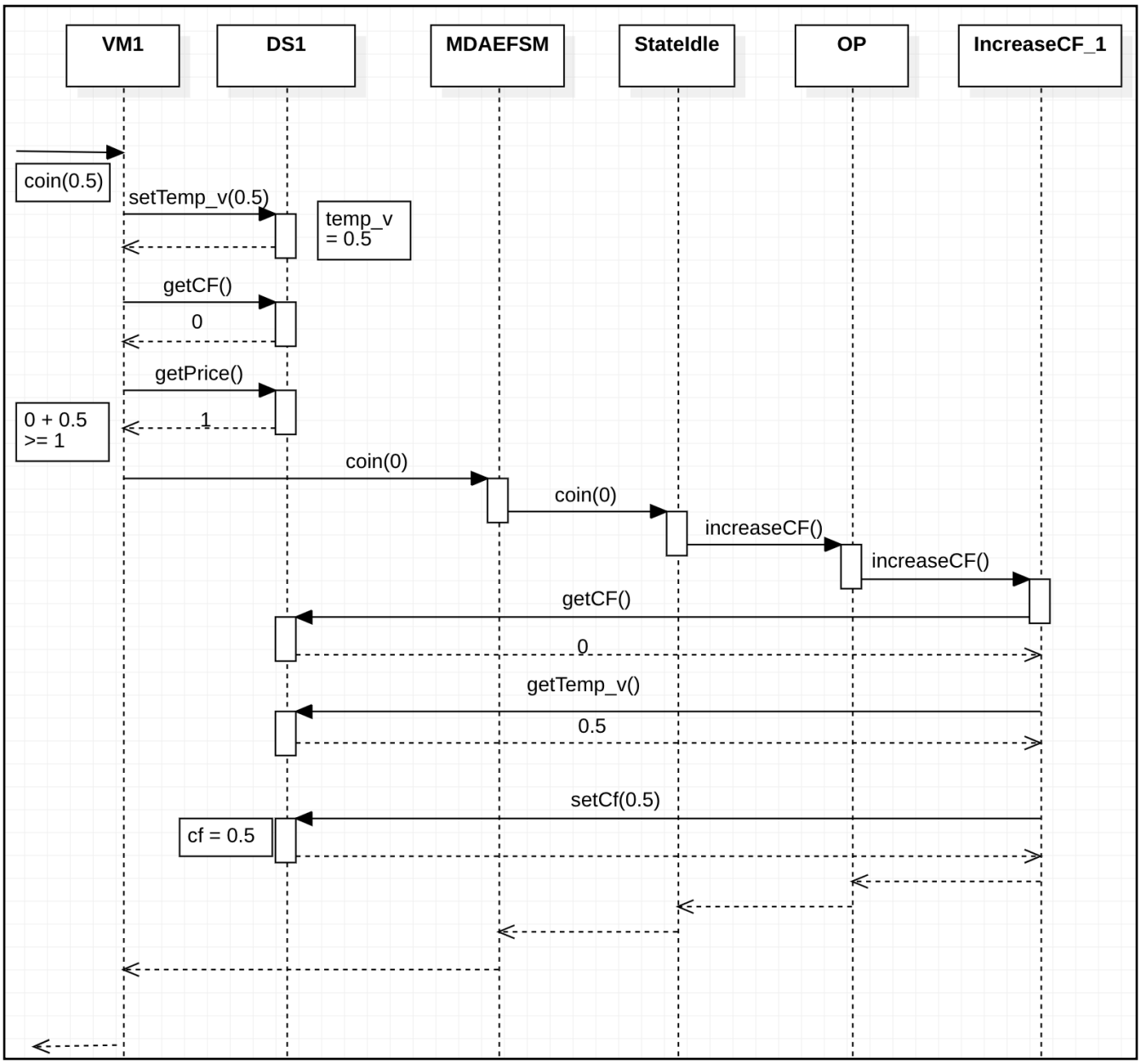
create(1):



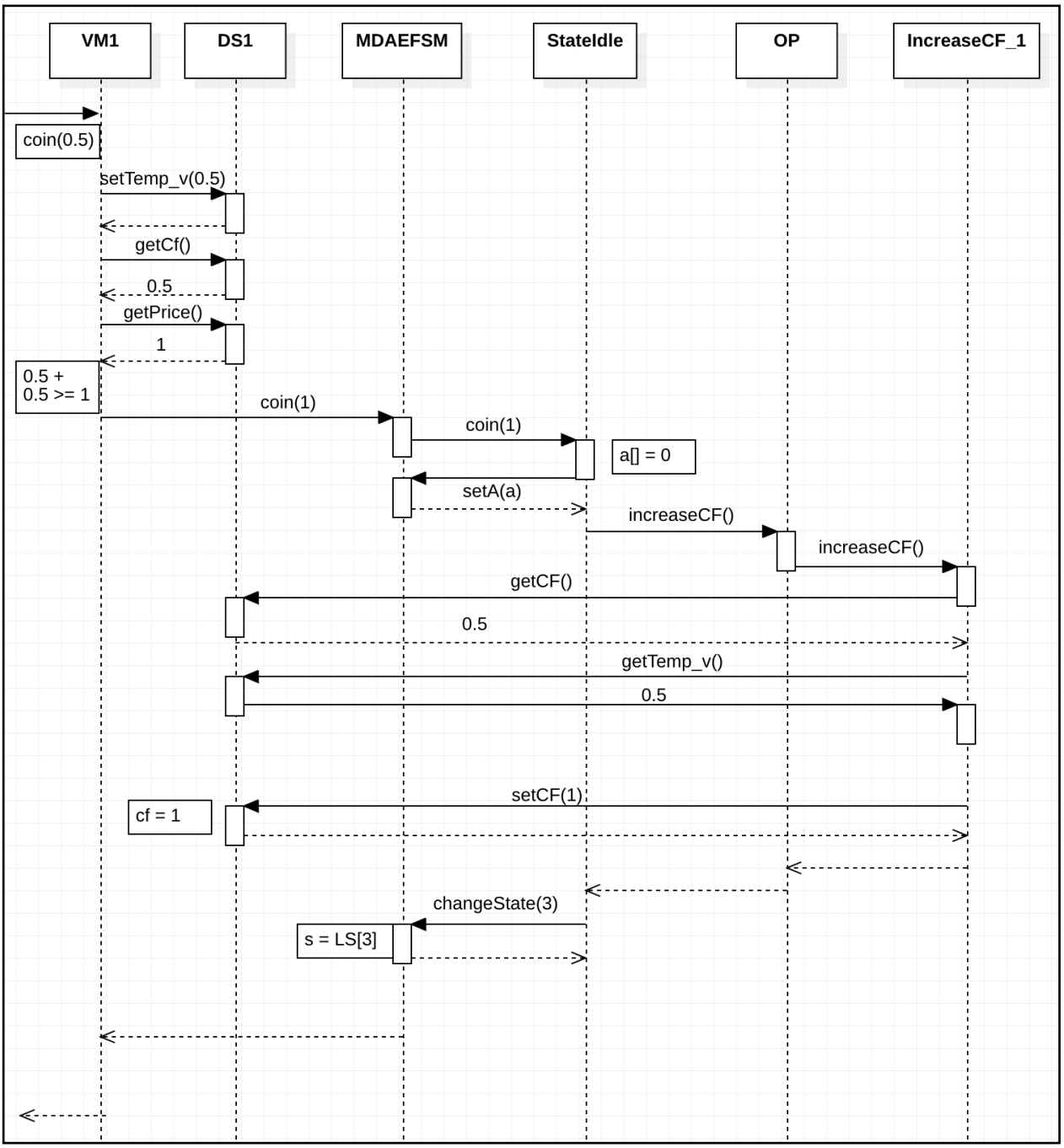
insert_cups(20):



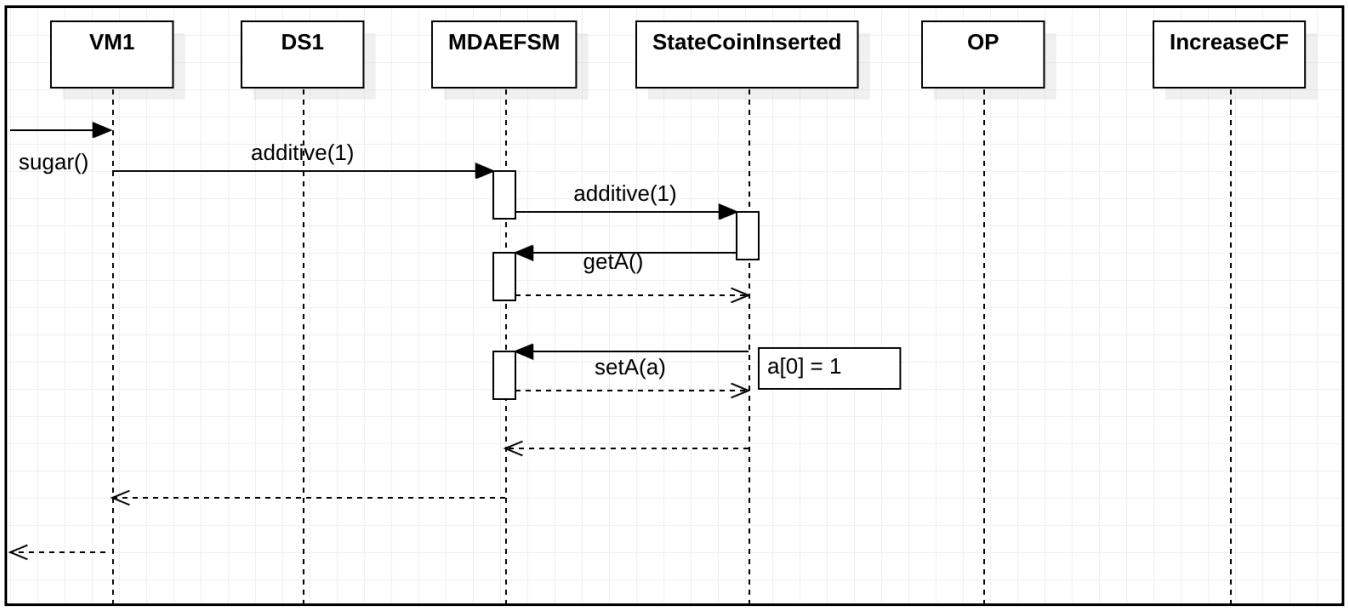
coin(0.5):


VM1
DS1
MDAEFSM
StateIdle
OP
IncreaseCF_1
coin(0.5)
setTemp_v(0.5)
temp_v
= 0.5
getCF()
0
getPrice()
0 + 0.5
>= 1
1
coin(0)
coin(0)
increaseCF()
increaseCF()
getCF()
0
getTemp_v()
0.5
setCf(0.5)
cf = 0.5

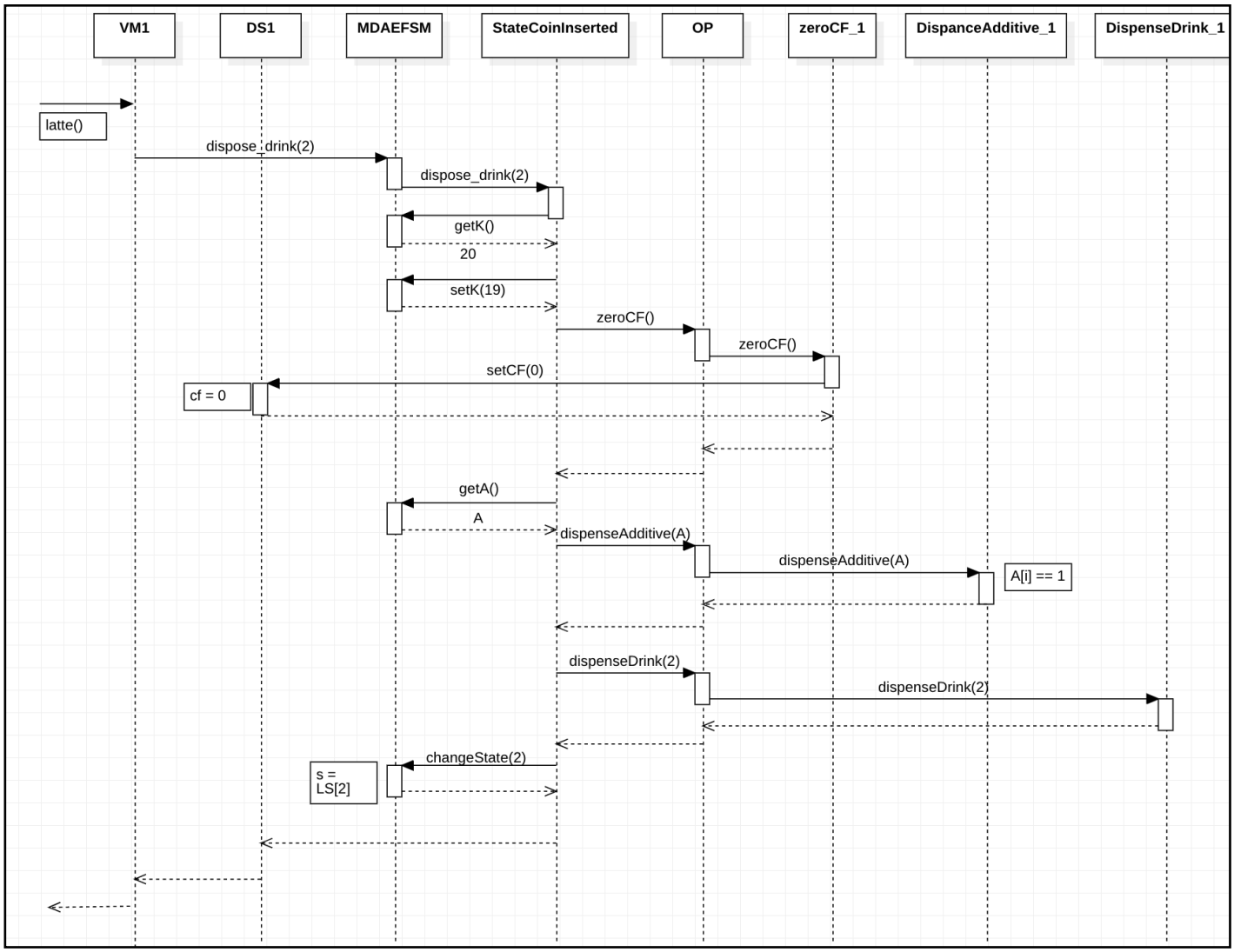
coin(0.5):


VM1
DS1
MDAEFSM
StateIdle
OP
IncreaseCF_1
coin(0.5)
setTemp_v(0.5)
getCf()
0.5
getPrice()
1
0.5 +
0.5 >= 1
coin(1)
coin(1)
a[] = 0
setA(a)
increaseCF()
increaseCF()
getCF()
0.5
getTemp_v()
0.5
setCF(1)
cf = 1
changeState(3)
s = LS[3]

sugar():


VM1
DS1
MDAEFSM
StateCoinInserted
OP
IncreaseCF
sugar()
additive(1)
additive(1)
getA()
setA(a)
a[0] = 1

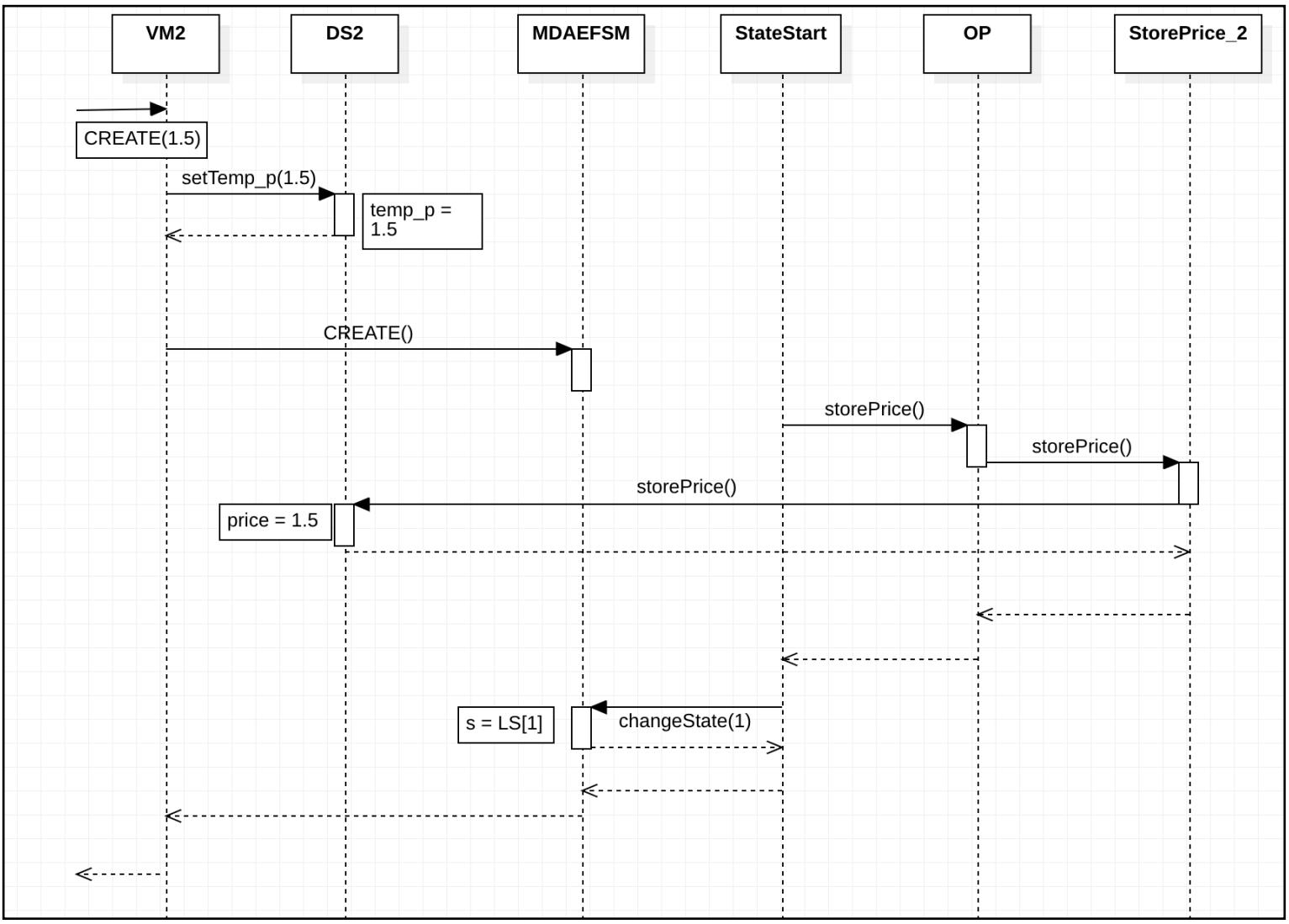
latte():


VM1
DS1
MDAEFSM
StateCoinInserted
OP
zeroCF_1
DispanceAdditive_1
DispenseDrink_1
latte()
dispose_drink(2)
dispose_drink(2)
getK()
20
setK(19)
zeroCF()
zeroCF()
setCF(0)
cf = 0
getA()
A
dispenseAdditive(A)
dispenseAdditive(A)
A[i] == 1
dispenseDrink(2)
dispenseDrink(2)
changeState(2)
s =
LS[2]

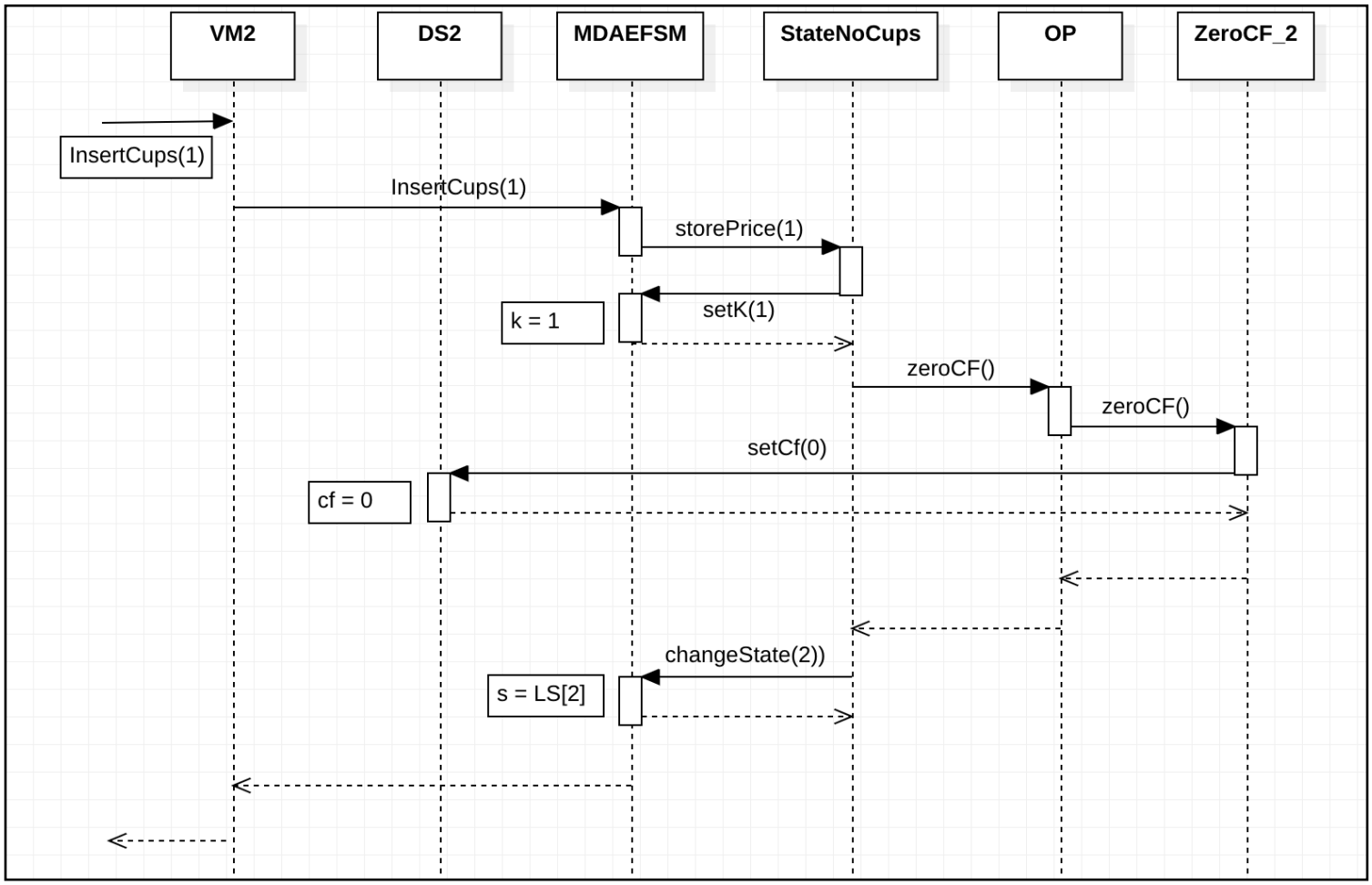
b. Vending machine 2 system sequence diagrams:

Note: Individual operations of the sequence diagram are added below. 1 single sequence diagram was hard to read.

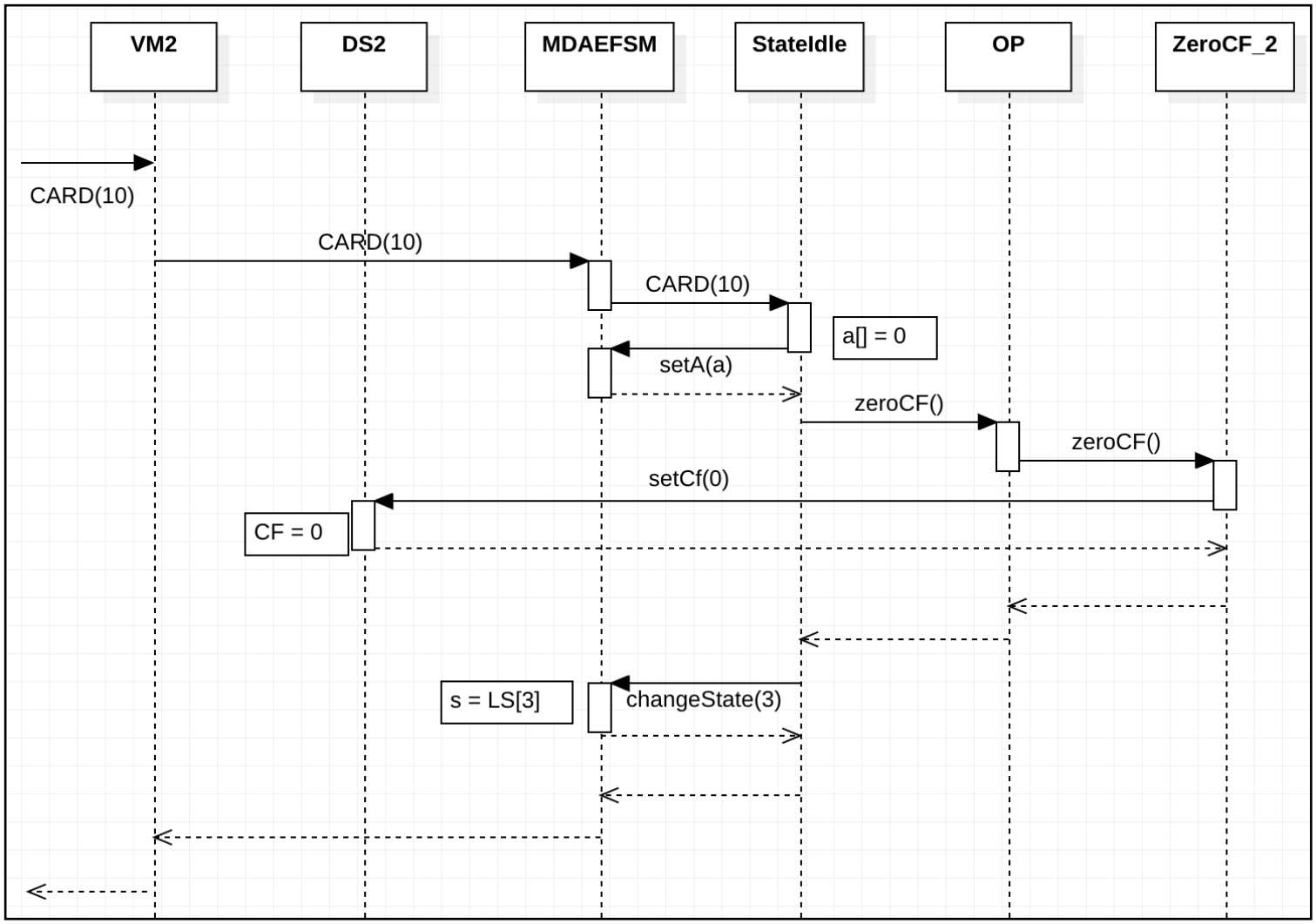
CREATE(1.5):



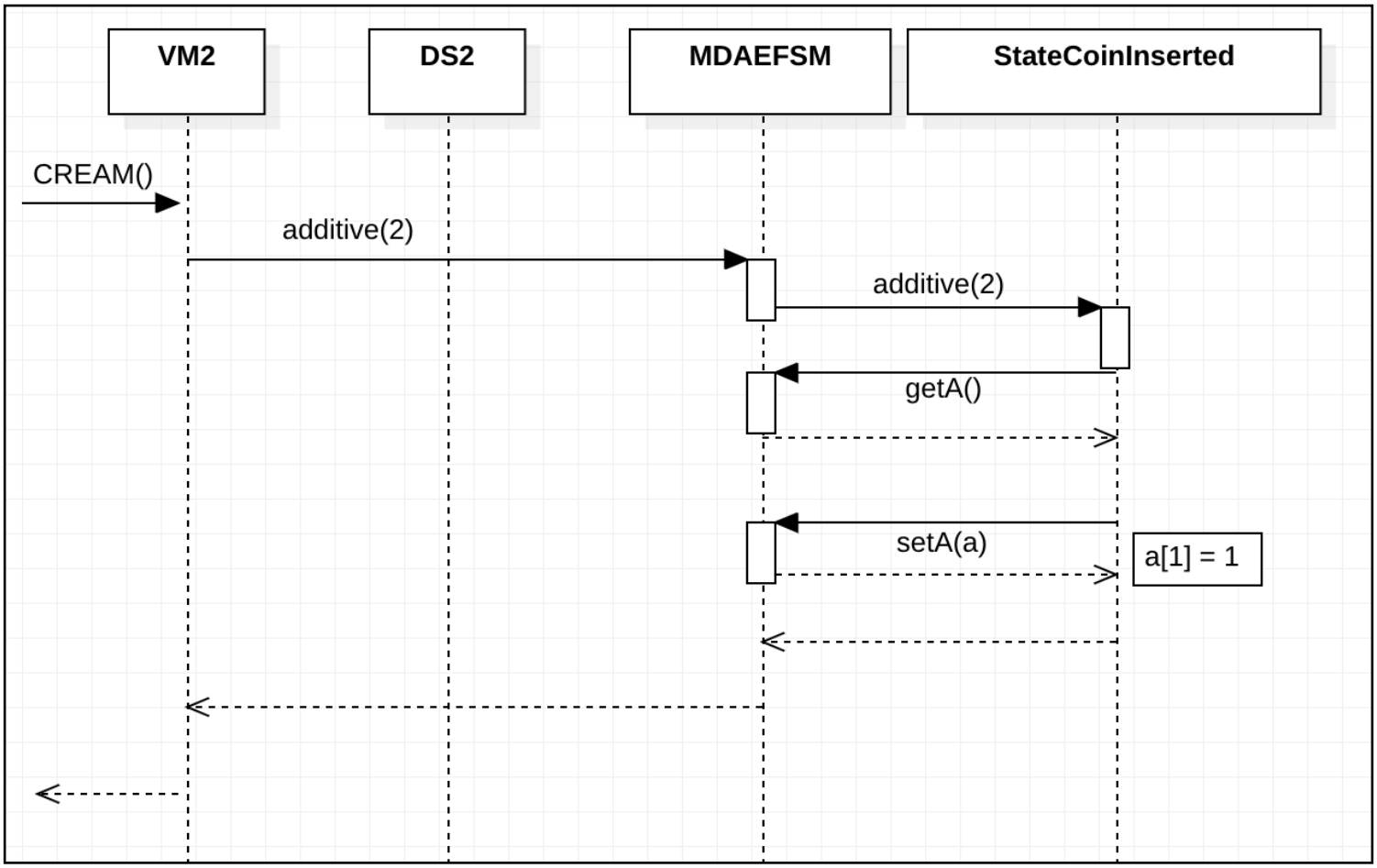
InsertCups(1):



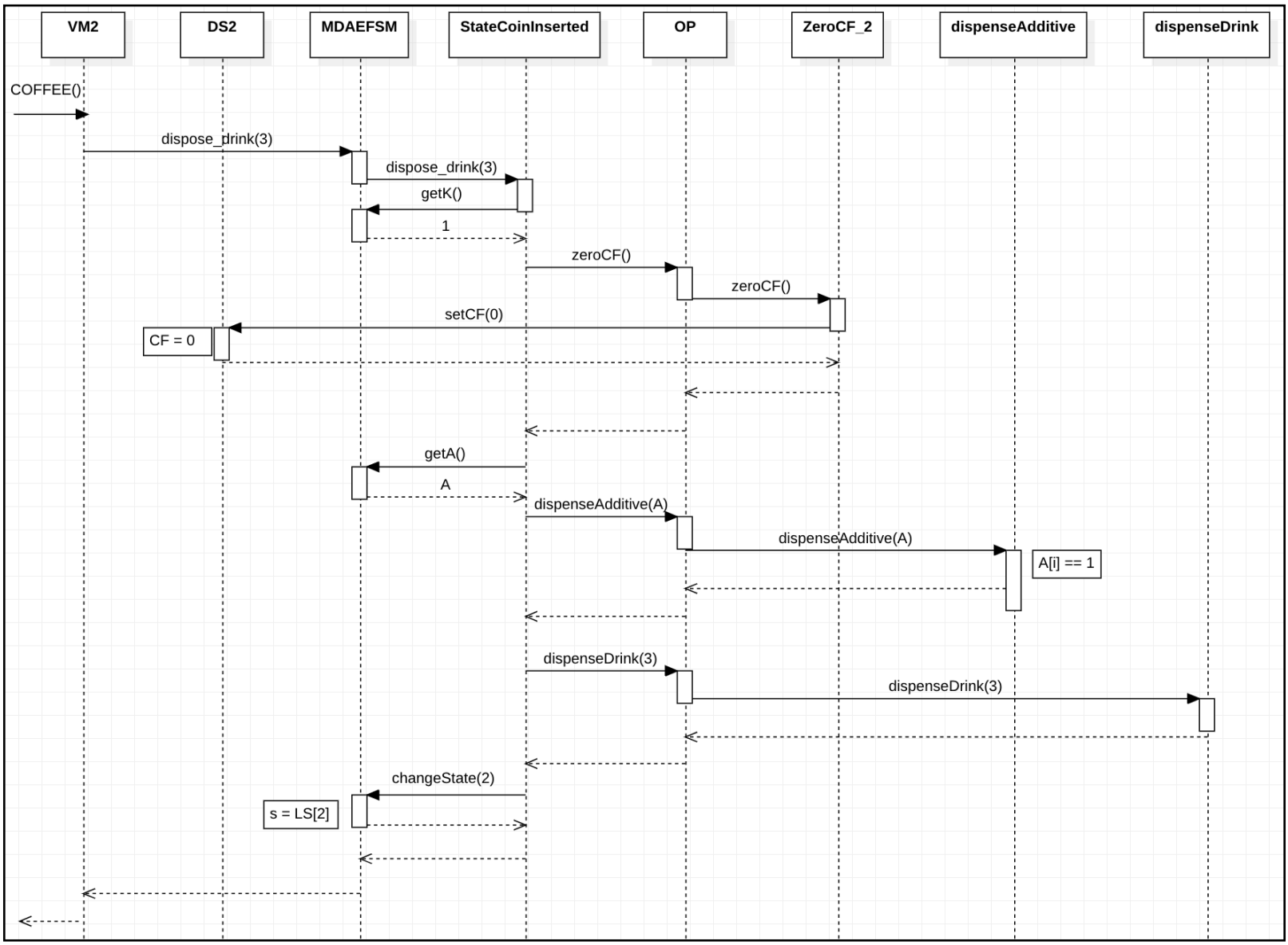
CARD(10):


VM2
DS2
MDAEFSM
StateIdle
OP
ZeroCF_2
CARD(10)
CARD(10)
CARD(10)
a[] = 0
setA(a)
zeroCF()
zeroCF()
setCf(0)
CF = 0
s = LS[3]
changeState(3)

CREAM():


VM2
DS2
MDAEFSM
StateCoinInserted
CREAM()
additive(2)
additive(2)
getA()
setA(a)
a[1] = 1

COFFEE():

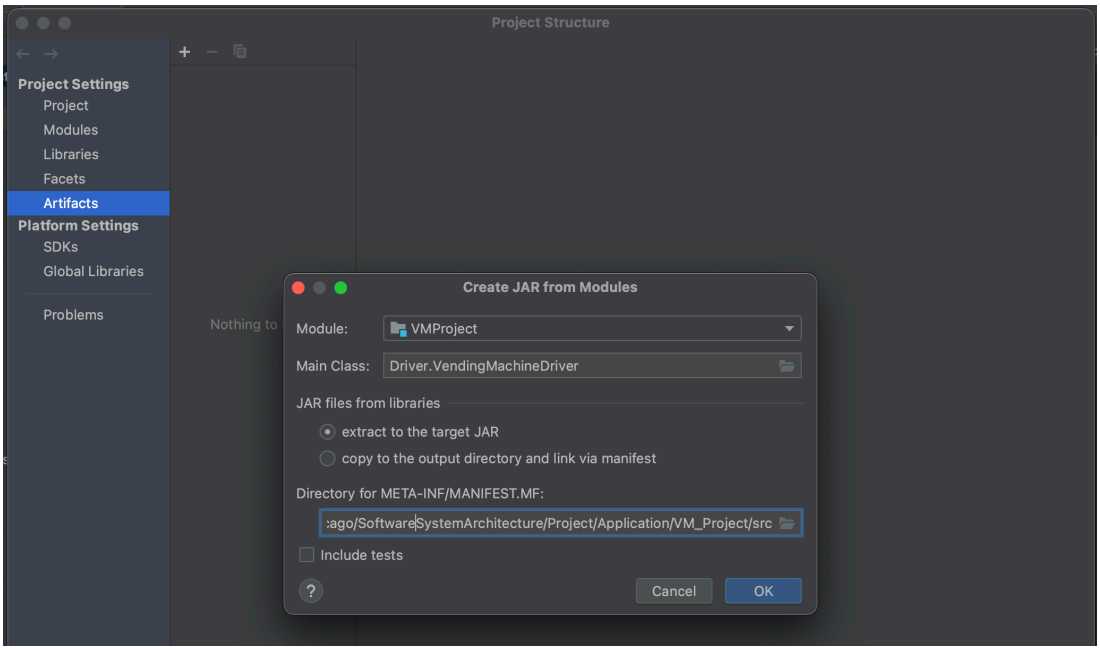

VM2
DS2
MDAEFSM
StateCoinInserted
OP
ZeroCF_2
dispenseAdditive
dispenseDrink
COFFEE()
dispose_drink(3)
dispose_drink(3)
getK()
1
zeroCF()
zeroCF()
setCF(0)
CF = 0
getA()
A
dispenseAdditive(A)
dispenseAdditive(A)
A[i] == 1
dispenseDrink(3)
dispenseDrink(3)
changeState(2)
s = LS[2]

About source code:

- The implementation of Vending Machine system is done in Java.
  - Java Version: Oracle OpenJDK Version(1.8.0_371)

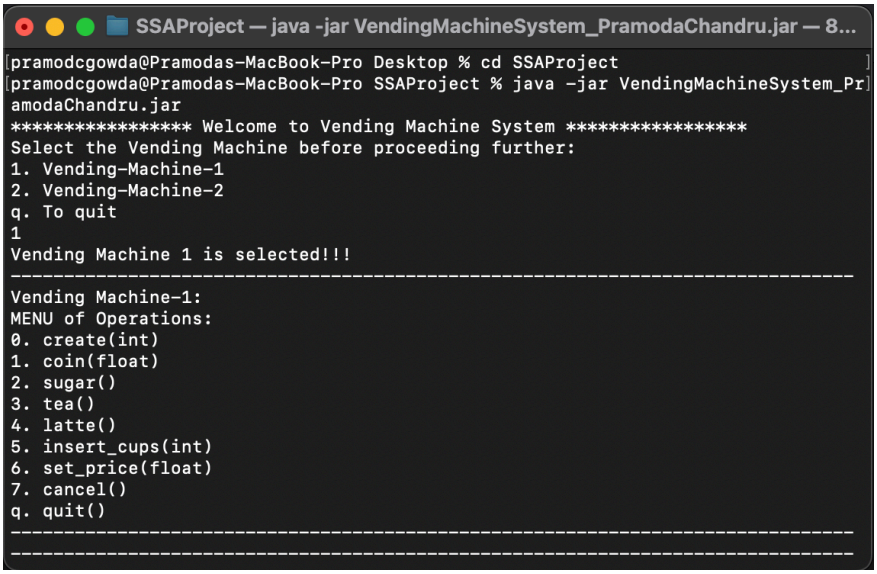
Build:

- Set the artifacts pointing to the main driver class
- Build the project to generate the Jar file.



Run:

- Open terminal
- Cd to the directory where the jar exists
- Run the command java -jar



- There is a total of 8 main packages created in the implementation
- The description of the packages is provided below:
  - DataStore: The classes created inside this package belong to the data store operation.
  - Driver: The classes created inside this package belong to the main class
  - **Factory: The classes created inside this package belong to the AbstractFactory Design Pattern.**
  - MDAEFSM: The classes created inside this package belong to the context class.
  - OutputProcessor: The classes created inside this package belong to the output operation
  - **State: The classes created inside this package belong to the State Design Pattern.**
  - **Strategy: The classes created inside this package belongs to the Strategy Design Pattern.**
  - VendingMachine: The classes created inside this package belong to Input Processor for the MDA Architecture.